EXHIBIT C

Samsung Galaxy S7

[Amended Claim Chart of February 22, 2018]

The Samsung Galaxy S7, discussed in the charts below, infringes U.S. Patent Nos. 6,429,846, 7,969,288, 7,982,720, 8,031,181, and 9,323,332.

A representative embodiment of the Samsung Galaxy S7 comprises the following hardware components for performing haptic feedback: a touch screen, which comprises a cover glass, capacitive sensor, touch controller to detect changes in capacitance at various nodes of the sensor when the surface is contacted by a user, display, and associated mechanical and electronic components; memory (including both RAM and flash); an applications processor, memory, and I/O; version 6.0 (Marshmallow) of the Android OS along with display and vibrator drivers; a power management chip; an actuator; and circuitry and components that ensure interoperability.

As an exemplary overview of signal flow, a touch on the cover glass results in a change in capacitance in the touch sensor that is detected by the touch controller. The touch controller provides position information (regarding the location of the touch in two dimensions, X and Y) to the applications processor. A touch screen driver running on the main processor translates these signals into a standard input format for Android, so they can be provided to the appropriate portions of the Android framework or Android applications. These can use Android software methods, such as View.performHapticFeedback() and Vibrator.vibrate(), to produce a haptic effect as a result of the information provided by the touch controller. Requests to perform haptic feedback passed through Android's Vibrator API are translated by an actuator driver into device-specific signals, which are used to control the actuator, such as through supplying voltage to the actuator and varying the voltage at appropriate time intervals. (*See generally* https://developer.android.com (containing reference guides for developers discussing these software methods).)

In this preface to Immersion's claim by claim analysis below, Immersion provides a non-exclusive set of exemplary infringing software implementations using this platform. The details of these exemplary scenarios are presented without prejudice to Immersion presenting additional infringing implementations based on other touch events or haptic responses. The exemplary implementations below, and examples within the charts, are based on publicly available information about stock Android version 7.0. [1] On information and belief, Samsung's phones operate or are capable of operating in a substantially similar manner, but Immersion has not yet had the

[1] Because the code in stock Android for certain touch events and haptic effects is substantially similar between Android 5.1 and Android 7.1, and all intervening versions, any citations in this preface and the charts below to Android 7.0 should be considered representative of the other versions of Android employed by phones at issue in this case.

opportunity to inspect Samsung's source code to confirm this. Immersion reserves its right to supplement these contentions once it has had the opportunity to analyze Samsung's source code. As Immersion receives discovery in this case, Immersion reserves the right to supplement the responses in this chart to include additional details, including relevant deposition testimony on these and any additional potentially relevant infringing implementations.

I. Touch Input Exemplary Implementation

- In stock Android, the View class represents one of the basic building blocks for user interface components. Touch input can be handled using standard Android input mechanisms where touch signals from the user are transmitted through device drivers, kernel input handlers, the Java Native Interface (JNI), Android system services, and to the View class hierarchy. This functionality is described by the documentation for the Android Open Source Project (AOSP):

- "Input Pipeline: At the lowest layer, the physical input device produces signals that describe state changes such as key presses and touch contact points. The device firmware encodes and transmits these signals in some way such as by sending USB HID reports to the system or by producing interrupts on an I2C bus. The signals are then decoded by a device driver in the Linux kernel. The Linux kernel provides drivers for many standard peripherals, particularly those that adhere to the HID protocol. However, an OEM must often provide custom drivers for embedded devices that are tightly integrated into the system at a low-level, such as touch screens. The input device drivers are responsible for translating device-specific signals into a standard input event format, by way of the Linux input protocol. The Linux input protocol defines a standard set of event types and codes in the linux/input.h kernel header file. In this way, components outside the kernel do not need to care about the details such as physical scan codes, HID usages, I2C messages, GPIO pins, and the like. Next, the Android EventHub component reads input events from the kernel by opening the evdev driver associated with each input device. The Android InputReader component then decodes the input events according to the device class and produces a stream of Android input events. As part of this process, the Linux input protocol event codes are translated into Android event codes according to the input device configuration, keyboard layout files, and various mapping tables. Finally, the InputReader sends input events to the InputDispatcher which forwards them to the appropriate window." (Ex. 1 at 1.)

II. Haptic Effects Exemplary Implementations

- Stock Android has an android.os package, which includes a Vibrator class. In stock Android, the Vibrator class (or its subclasses) implements the method vibrate(long[] pattern, int repeat) for vibrating with a given pattern and the method vibrate (long milliseconds) for vibrating for a specified time period. (Ex. 2.) Stock Android also has an android.view package, which

includes a View class. In stock Android, the View class (or its subclasses) implements the method performHapticFeedback(int feedbackConstant) and the method performHapticFeedback(int feedbackConstant, int flags) for providing haptic feedback to the user for a View object. For example, when a user performs a touch event within a View object, the method onTouch(View v, MotionEvent event) and/or the method onTouchEvent(MotionEvent event) may cause a processor (computer device) to call these performHapticFeedback methods and output information or signals to provide a haptic effect, if the global setting HAPTIC_FEEDBACK_ENABLED is True or the method is called with the flag FLAG_IGNORE_GLOBAL_SETTING as a parameter. (Ex. 3.)

- Additionally, on information and belief, in many of Samsung's accused instrumentalities, the Vibrator class (or its subclass SystemVibrator) implements a Samsung-added custom method semVibrate that causes the accused instrumentalities to vibrate when called with certain parameters that represent the force to be output by the actuator in the device. Immersion has not yet had the opportunity to analyze Samsung's source code or take other discovery about Samsung's custom method semVibrate, and reserves the right to supplement these charts and its contentions as Immersion learns more through discovery.

A. Virtual Key, Keyboard, and Long Press Examples

- In one exemplary implementation of virtual keys, stock Android receives information from the touchscreen that a user has touched a virtual key (*e.g.*, Back or Recent). The phone will then vibrate in response to such a touch. In this example, user input on a virtual key is received by the Android input system and is handled by the PhoneWindowManager class in method interceptKeyBeforeQueueing(KeyEvent event, int policyFlags). (Ex. 9.) This method then calls performHapticFeedbackLw() with the HapticFeedbackConstant for a VIRTUAL_KEY as a parameter. The performHapticFeedbackLw() method then calls one of the two vibrate() methods in the Vibrator class, if the global setting HAPTIC_FEEDBACK_ENABLED is True or the method is called with the flag FLAG_IGNORE_GLOBAL_SETTING as a parameter. (Exs. 2, 9.)

- In another exemplary implementation, stock Android handles a user touch on a keyboard key as follows: Touches on the keyboard interface are received by method onPressKey() of class LatinIME. (Ex. 12.) Method onPressKey() then calls method performHapticFeedback() of class AudioAndHapticFeedbackManager (Ex. 13), which then calls method performHapticFeedback() of class View, and passes in the HapticFeedbackConstant KEYBOARD_TAP as a parameter. (Exs. 3, 10.) As described above, this results in a call to one of the vibrate() methods of class Vibrator, if the global setting HAPTIC_FEEDBACK_ENABLED is True or the method is called with the flag FLAG_IGNORE_GLOBAL_SETTING as a parameter. (Ex. 2.)

- Another example occurs when the user long presses on a list view. In this example, the performLongPress(final View child, final int longPressPosition, final long longPressId) method in the AbsListView class determines if any of the items has consumed the long press. (Ex. 14.) If so, the Android framework calls method performHapticFeedback(HapticFeedbackConstants.LONG_PRESS) in class View to cause the actuator to impart a force. (Exs. 3, 14.) As described above, this results in a call to one of the vibrate() methods of class Vibrator, if the global setting HAPTIC_FEEDBACK_ENABLED is True or the method is called with the flag FLAG_IGNORE_GLOBAL_SETTING as a parameter. (Ex. 2.)

B. Framework and Kernel Code

- When an application initiates haptic feedback, the application calls the vibrate() method of the Vibrator class, either indirectly through the View.performHapticFeedback() method or with a direct call to Vibrator.vibrate(). Regardless of whether Vibrator.vibrate() is called directly by an application or through the View.performHapticFeedback() method, Vibrator.vibrate() may be called with two types of arguments. Vibrator.vibrate() may be called with a duration value that defines the duration of the haptic feedback (*i.e.*, how long the vibrator should provide force feedback). Vibrator.vibrate() may also be called with an array of values defining a vibration pattern and a parameter determining whether any portion of the vibration pattern is to be repeated. In the array, "the first value indicates the number of milliseconds to wait before turning the vibrator on. The next value indicates the number of milliseconds for which to keep the vibrator on before turning it off. Subsequent values alternate between durations in milliseconds to turn the vibrator off or to turn the vibrator on." (Ex. 15.) The repeat parameter determines whether the vibration pattern will be repeated and, if so, at which array index to start the repeat. (Ex. 2.) The Vibrator class also provides versions of these two methods that include an extra parameter (AudioAttributes attributes) that allows a developer to include tags that indicate the reason for the vibration and the type of vibration.

- When the vibrate method is called with either type of argument, the Vibrator class uses the Android VibratorService to pass the requested haptic feedback to Android's Hardware Abstraction Layer (HAL) and then to the vibrator driver of the Linux kernel. (Ex. 16.) In the case where Vibrator.vibrate() is called with a duration parameter, the duration is passed to the VibratorService and on to the HAL using JNI function vibratorOn(long milliseconds). (Ex. 17.) The HAL then passes the information to the vibrator driver using the "sysfs" virtual filesystem generated by the Linux kernel. (Ex. 18.) The kernel then reads this duration parameter and activates the vibrator for the duration set in the duration parameter.

- In the case where Vibrator.vibrate() is called with a vibration pattern and a repeat parameter, the vibratePattern() method of the VibratorService passes appropriately spaced duration values to the HAL and thus to the kernel to create the requested vibration pattern. Specifically, the vibratePattern() method will wait for an amount of time according to the first value of the vibration